Compas'2016 : Parallélisme / Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Problème d'ordonnancement dans Hadoop, ordonnancement sur des machines parallèles équivalentes et distribuées

Aymen Jlassi [◊], Patrick Martineau[◊]

Université François-Rabelais de Tours,[◊]
CNRS, LI EA 6300, OC ERL CNRS 6305, Tours, France
64 Avenue Jean de Portalis, 37200 Tours - France
{aymen.jlassi, patrick.martineau}@univ-tours.fr

Résumé
Les outils de gestion de gros volumes de données sont connus pour leur capacité à exécuter un grand nombre de travaux sur des volumes de données énormes (de l'ordre de plusieurs Pétaoctets). De ce fait, ces outils utilisent de grandes infrastructures capables de fournir la puissance de calcul demandée de manière à effectuer les traitements dans un temps raisonnable.
Dans ce papier, on s'intéresse à l'amélioration des performances du logiciel "Hadoop", le logiciel libre de référence dans l'univers des logiciels de traitement de gros volumes de données. Dans une première partie, on modélise le problème d'ordonnancement à l'aide de la programmation linéaire, on évalue le modèle et on calcule ainsi une borne inférieure pour la version hors ligne du problème. On propose une heuristique et on évalue la solution qu'on propose sur des instances de taille moyenne.

Mots-clés : Architecture parallèle et distribuée, Ordonnancement, Grid & Clouds

---

1. Introduction

Suite à la publication des travaux de recherche chez Google en 2004 : la présentation du modèle MapReduce [2] et les fondements d'un système de fichier distribué GFS, Yahoo ! a créé Hadoop, un logiciel libre capable de traiter des énormes quantités de données et de répondre aux besoins de plusieurs entreprises. Hadoop est basé sur une architecture client-serveur, l'utilisateur soumet les travaux au serveur et ce dernier s'occupe de leur ordonnancement sur la grappe. De nos jours, Facebook, General motors, Orange et beaucoup d'autres entreprises utilisent Hadoop dans leurs centres de données.
Notre objectif est l'optimisation des performances de Hadoop en améliorant la politique d'ordonnancement des travaux sur la grappe. Il existe plusieurs références bibliographiques abordant le problème d'ordonnancement des travaux Map/Reduce dans le contexte du logiciel Hadoop. Le premier ordonnanceur dans Hadoop est FIFO (premier arrivé premier servi), cet ordonnanceur exécute les travaux dans l'ordre de leur soumission. "Fair" [7], "capacity" [3] et "Dynamic Priority scheduler" [6] sont des ordonnanceurs hiérarchiques constitués de plusieurs niveaux de files d'attente avec des priorités entre elles. "Fair" résout le problème de famine rencontré en utilisant FIFO et "Capacity" gère davantage l'hiérarchie de files d'attente que "Fair".

Pour plus de détails concernant les travaux de recherche abordant l'ordonnancement dans Hadoop, les papiers [5], [1] et [4] synthétisent la plupart de ces travaux.
On définit le problème d'ordonnancement dans Hadoop comme suit : on considère une grappe constituée de M machines. Chaque machine M_j est caractérisée par (1) un ensemble de slots m_j^s dont m_j^{Sm} slots qui exécutent des tâches Map et m_j^{Sr} slots qui exécutent des tâches Reduce, (2) une capacité m_j^r de mémoire (RAM) et une quantité m_j^h de disque dur (3) une fréquence de calcul v_j de chaque slot. On considère un ensemble de travaux W de types Map / Reduce et un ensemble de blocs de données noté A^b. Un travail est un ensemble de tâches Map et de tâches Reduce. Ces travaux sont décomposés en un ensemble de $N(= N^m + N^r)$ tâches (composés de deux sous ensembles : L^m tâches Map et L^r tâches Reduce) dont $N^m(= card(L^m))$ tâches Map et $N^r(= card(L^r))$ tâches Reduce. Chaque tâche $T_i^{w'}$ est associée à un travail $w' \in W$ et sa durée d'exécution est notée p_i. On note $b_{i,i'}$ la bande passante réservée pour assurer l'échange des données entre deux tâches $T_i^{w'}$ et $T_{i'}^{w'}$ qui s'exécutent sur deux machines différentes. Pour chaque tâche $T_i^{w'}$, on considère E_i l'ensemble des n_i^p tâches qui doivent s'exécuter avant la tâche $T_i^{w'}$. Lorsque la tâche $T_i^{w'}$ s'exécute, elle nécessite une quantité n_i^r de RAM, une quantité n_i^h de disque dur, un slot et traite un ensemble B_i de données divisées en $n_i^b \geqslant 1$ blocs lus sur le système de fichiers. Dans la version actuelle du logiciel, une tâche Map dans Hadoop effectue des traitements sur un seul bloc de données [8]. Tous les blocs de données ont la même taille S. On note r_b le nombre de réplications d'un bloc b. D_b est alors l'ensemble des machines qui contiennent le bloc b et bwd la bande passante allouée pour migrer un bloc d'une machine à une autre, si nécessaire. L'architecture de la grappe est modélisée par un graphe $G = (V, E)$ où l'ensemble des nœuds V présente les machines et l'ensemble des arcs E présente les liaisons filaires entre elles. On considère P l'ensemble des chemins entre les machines, P_u est l'ensemble des couples de machines qui utilisent l'arc u. Chaque chemin est composé de plusieurs arcs $e_u \in E$ dont la capacité maximale de bande passante bwd est identique pour tous les arcs. On note T l'horizon de temps nécessaire pour l'ordonnancement des tâches sur les machines en tenant compte de la gestion des ressources. On cherche à minimiser la somme des dates de fins des exécutions des travaux.
Dans une première partie, on traite le problème dans sa version hors ligne, on le modélise mathématiquement, on évalue le modèle et on calcule une borne inférieure. Dans une deuxième étape, on propose une heuristique de résolution et on l'évalue par rapport à la borne inférieure trouvée précédemment.

2. Modélisation mathématique du problème

Dans cette partie, on présente une formulation mathématique basée sur la programmation linéaire, indexée par le temps du problème d'ordonnancement. Les variables du modèle sont présentées dans (1), (2), (3), (4) et (5).

$$x_{i,t}^{j,s} = \begin{cases} 1 & \text{si la tâche i est exécutée sur le slot s de la machine j à l'instant (t, t+1]} \\ 0 & \text{Sinon} \end{cases} \quad (1)$$

$$y_{b,t}^{j,j'} = \begin{cases} 1 & \text{si le bloc b est sur la machine j à l'instant } (t, t+1] \text{ après une migration} \\ & \quad \text{de la machine j' avant l'instant t} \\ 0 & \text{Sinon} \end{cases} \quad (2)$$

$$u_{b,t}^{j,j'} = \begin{cases} 1 & \text{si le bloc b est en cours de migration de la machine j vers la machine j'} \\ & \quad \text{à l'instant (t, t+1]} \\ 0 & \text{Sinon} \end{cases} \quad (3)$$

$$z^{j,j'}_{l,l',t} = \begin{cases} 1 & \text{si la tâche Map l' a terminé son exécution à l'instant t sur la machine j'} \\ & \quad \text{et la tâche l est exécutée sur la machine j après l'instant t} \\ 0 & \text{Sinon} \end{cases} \quad (4)$$

$$C_{w'} = t, \forall w' = 1...W, t \in [1..T] : \text{ Date de fin d'exécution de la dernière tâche du travail w'.} \quad (5)$$

On cherche à minimiser la somme des dates de fin des différents travaux soumis. La fonction objective est :

$$MinimizeObj = \sum_{w'=1}^{W} C_{w'}$$

On a quatre types de contraintes :
Les contraintes de ressources : La contrainte (6) suppose qu'à chaque instant, la quantité de mémoire consommée par les tâches qui s'exécutent sur une machine j ne dépasse pas la quantité de mémoire sur la machine. Les contraintes (7) et (8) imposent qu'un slot (coeur virtuel) ne puisse exécuter qu'une seule tâche à la fois. La contrainte (9) contrôle l'utilisation du disque dur. À chaque instant, la somme de la quantité du disque dur (HDD) utilisée par les tâches et la quantité de HDD utilisée pour l'enregistrement et la migration des données ne doivent pas dépasser la quantité disponible sur la machine.
Les contraintes reliées à l'exécution des tâches : La contrainte (10) sert à définir la date de fin de la dernière tâche de chaque travail. La contrainte (11) définit la relation de précédence entre les tâches. Les contraintes (**??**) et (15) définissent les durées des tâches "Map" et "Reduce". Les contraintes (12) et (16) imposent qu'à chaque instant, une tâche "Map" ou "Reduce" ne puisse s'exécuter que sur un seul slot. Les contraintes (13), (14), (17) et (18) interdisent l'interruption des tâches.
Les contraintes reliées à la migration et le transfert des données : Les contraintes (19) et (20) définissent si un bloc de données est enregistré sur une machine ou non. Les contraintes (21) et (22) imposent qu'une tâche "Map" ne puisse s'exécuter sur une machine que si les données, qu'elle traite, y sont présentes. Les contraintes (23) et (24) forcent la suppression d'un bloc, déjà migré, lorsqu'il n'existe plus de tâche qui en a besoin sur la machine où il a été migré. La contrainte (24) interdit la migration d'un bloc s'il est déjà présent sur la machine de destination.
Les contraintes reliées à la gestion du réseau : La contrainte (25) impose que la quantité de données transférées à travers le réseau à un instant donné, pour la migration de données et l'échange de données entre les tâches, ne dépasse pas la bande passante disponible sur les arcs. La contrainte (26) permet la réservation de la bande passante depuis la fin de la tâche "Map" jusqu'à la fin des "Reduce" associés.
Après la modélisation mathématique du problème, l'étape suivante focalise sur l'évaluation du modèle et le calcul d'une borne inférieure.

$$\sum_{s=1}^{m^s_j} \sum_{i=1}^{N} n^r_i x^{j,s}_{i,t} \leq m^r_j. \quad \forall j = 1...M, \forall t = 1...T \quad (6)$$

$$\sum_{i \in L^r} x^{j,s}_{i,t} \leq 1. \quad \forall j = 1...M, \forall t = 1...T, \forall s = 1...m^{Sr}_j \quad (7)$$

$$\sum_{i \in L^m} x^{j,m^{Sr}_j+s}_{i,t} \leq 1. \quad \forall j = 1...M, \forall t = 1...T, \forall s = 1...m^{Sm}_j \quad (8)$$

$$\sum_{s=1}^{m^s_j} \sum_{i=1}^{N} n^h_i x^{j,s}_{i,t} + \sum_{j'=1, j \neq j'}^{M} \sum_{i=1}^{N} \sum_{b \in B_i} S(y^{j,j'}_{b,t} + u^{j,j'}_{b,t}) \leq m^h_j. \quad \forall j = 1...M, \forall t = 1...T \quad (9)$$

$$\sum_{j=1}^{M} \sum_{s=1}^{m^{sr}_j} (t+1) x^{j,s}_{i,t} \leq C_{w'}. \quad \forall w\prime = 1...W, \forall t = 1...T, \forall i \in L^r \cup \{l\} \text{ tel que } l \in w' \quad (10)$$

$$\left(\sum_{l\in E_k} p_l\right) * x_{k,t}^{j,s} \leq \sum_{u=1}^{M}\sum_{s'=1}^{m_j^{S_m}}\sum_{t'=1}^{t}\sum_{l\in E_k} x_{l,t'}^{u,m_u^{S_r}+s'}. \quad \forall k \in L_r, \forall t = 1...T, \forall j = 1...M, \forall s = 1...m_j^{S_r} \quad (11)$$

$$\sum_{j=1}^{M}\sum_{s=1}^{m_j^{S_m}}\sum_{t=1}^{T} x_{i,t}^{j,m_j^{S_r}+s} = P_i. \quad \forall i \in L^m \quad (12)$$

$$\sum_{j=1}^{M}\sum_{s=1}^{m_j^{S_m}} x_{i,t}^{j,m_j^{S_r}+s} \leq 1. \quad \forall i \in L^m, \forall t = 1...T \quad (13)$$

$$p_i - (1 - x_{i,t}^{j,m_j^{S_r}+s} + x_{i,t-1}^{j,m_j^{S_r}+s}) * HV \leq \sum_{t'=t}^{t+p_i} x_{i,t'}^{j,m_j^{S_r}+s} \quad (14)$$

$$\forall i \in L^m, \forall t = 2...T - p_i, \forall j = 1...M, \forall s = 1...m_j^{S_m}, (HV : \text{high value})$$

$$p_i - (1 - x_{i,0}^{j,m_j^{S_r}+s}) * HV \leq \sum_{t'=t}^{t+p_i} x_{i,t'}^{j,m_j^{S_r}+s}. \quad \forall i \in L^m, \forall t = 2...T - p_i, \forall j = 1...M, \forall s = 1...m_j^{S_m} \quad (15)$$

$$\sum_{j=1}^{M}\sum_{s=1}^{m_j^{S_r}}\sum_{t=1}^{T} x_{i,t}^{j,s} = P_i. \quad \forall i \in L^r \quad (16)$$

$$\sum_{j=1}^{M}\sum_{s=1}^{m_j^{S_r}} x_{i,t}^{j,s} \leq 1. \quad \forall i \in L^r, \forall t = 1...T \quad (17)$$

$$p_i - (1 - x_{i,t}^{j,s} + x_{i,t-1}^{j,s}) * HV \leq \sum_{t'=t}^{t+p_i} x_{i,t'}^{j,s} \quad (18)$$

$$\forall i \in L^r, \forall t = 2...T - p_i, \forall j = 1...M, \forall s = 1...m_j^{S_r}, (HV : \text{high value})$$

$$p_i - (1 - x_{i,0}^{j,s}) * HV \leq \sum_{t'=t}^{t+p_i} x_{i,t'}^{j,s}. \quad \forall i \in L^r, \forall t = 2...T - p_i, \forall j = 1...M, \forall s = 1...m_j^{S_r} \quad (19)$$

$$y_{b,t}^{j,j} = \begin{cases} 1 & \forall j \in D_b \\ 0 & \forall j \notin D_b \end{cases}. \quad \forall t = 1...T, \forall b \in A^b \quad (20)$$

$$y_{b,1}^{j,j'} = 0. \quad \forall b \in A^b, \forall j = 1...M, \forall j' = 1...M, j' \neq j \quad (21)$$

$$\sum_{s=1}^{m_j^{S_m}} x_{i,t}^{j,m_j^{S_r}+s} \leq \sum_{b\in b_i}\sum_{j'\in D_b} y_{b,t}^{j,j'}. \quad \forall i \in L^m, \forall j = 1...M, \forall t = 1...T \quad (22)$$

$$x_{i,t}^{j,m_j^{S_r}+s} \leq \sum_{j'\in D_b} y_{b,t-1}^{j,j'} + \sum_{j'\in D_b} u_{b,t-1}^{j,j'} \quad \forall i \in L^m, \forall b \in B_i, \forall j = 1...M, \forall s = 1...m_j^{S_m}, \forall t = 2...T \quad (23)$$

$$\sum_{j'\in D_b, j'\neq j} y_{b,t}^{j,j'} \leq \sum_{t'=t+1}^{T}\sum_{s=1}^{m_j^{S_m}} x_{i,t'}^{j,m_j^{S_r}+s}. \quad \forall i \in L^m, \forall b \in B_i, \forall j = 1...M, \forall t = 1...T \quad (24)$$

$$u_{b,t}^{j,j'} \leq 1 - y_{b,t}^{j,j'}. \quad \forall j = 1...M, \forall j' = 1...M, \forall b \in A^b, \forall t = 1...T \quad (25)$$

$$\sum_{(j,j')\in P_e}\left[bwd\sum_b u_{b,t}^{j,j'} + \sum_{i\in L^r}\sum_{l\in E_i}\sum_{s=1}^{m_j^s} z_{i,l,t}^{j,j'} * b_{i,l}\right] \leq b_{max}. \quad \forall e \in E, \forall t = 1...T \quad (26)$$

$$\sum_{s=1}^{m_j^{S_r}} x_{l,t''}^{j,s} + \sum_{s=1}^{m_{j'}^{S_m}} x_{l',t'}^{j,m_{j'}^{S_r}+s} - 1 \leq z_{l,l',t}^{j,j'} \quad (27)$$

$$\forall l \in L^r, \forall l' \in E_l, \forall t = 1...T-1, \forall t' = 1...t-1, \forall t'' = t...T, \forall j, j' = 1...M, j \neq j'$$

3. Évaluation du modèle

L'évaluation du modèle a pour objectif de déterminer la taille des problèmes pour lesquels il est utilisable, et pour lesquels il permettra de déterminer une solution exacte ou une solution approchée.

Génération des instances : les données d'entrée de notre modèle sont les données reliées aux machines, aux tâches, aux blocs et à la typologie réseaux. Comme il n'existe pas d'instances de référence pour ce problème, on génère des instances à l'aide d'un générateur aléatoire. On s'appuie sur les architectures proposées par le site d'Amazon AWS. Le tableau (1) affiche les trois configurations de machines utilisées, les colonnes présentent par ordre : la description de la catégorie, le nombre de cœurs (physique) de calcul, la quantité de mémoire, l'espace d'enregistrement sur les disques et la fréquence de calcul de chaque cœur. On a choisi de réserver un cœur (physique) CPU pour les opérations du système d'exploitation de chaque machine (hypothèse raisonnable en pratique). Les deux dernières colonnes présentent la quantité de cœurs virtuels de calcul (slots). On génère les informations reliées aux tâches en fonction de la taille des données qu'elles traitent et du type du travail (tableau (3)) auquel elles appartiennent. On considère trois types de travaux : (i) des travaux nommés "CPU-intensive" (type (1)), ils utilisent plus la ressource de calcul (slots). (ii) des travaux nommés "RAM-intensive" (type (2)) qui ont une forte utilisation de la mémoire RAM. (iii) des travaux nommés "I/O-intensive" (type (3)) qui ont une forte utilisation de l'espace disque. Durant les simulations, on génère aléatoirement les types et les tailles des travaux. En fonction du type du travail auquel la tâche appartient, on génère la quantité de mémoire RAM (n_i^r) et la quantité d'espace disque (n_i^h) nécessaire pour chaque tâche, on utilise les formules décrites dans le tableau (3). La durée d'exécution des tâches est générée aléatoirement ($p_i \in [1, 2, 3]$). On considère que les quantités de données d'entrée et de sortie d'une tâche Map sont égales et que la quantité de données d'entrée d'une tâche Reduce est égale à : $\frac{\text{la somme des données d'entrée de toutes les tâches "Map" qui la précèdent}}{\text{le nombre de tâches "Map" qui la précèdent}}$. Si deux tâches de type "Map" et "Reduce", d'un même travail, sont sur deux machines différentes, la bande passante allouée pour assurer le transfert de données entre les deux tâches est générée entre 30 et 100% de la taille de la donnée d'entrée de la tâche "Map". On considère que la taille S des blocs est égale à 64 Mb, la bande passante nécessaire à la migration de chaque bloc est $bwd = min[S*0.3, S]$. L'affectation des blocs (et leurs réplications) aux machines et l'affectation des blocs aux tâches sont générées aléatoirement. On utilise une architecture sous la forme d'un arbre dont les machines sont les feuilles et les nœuds intermédiaires présentes des routeurs informatiques. On génère les chemins entre les machines et les arcs qui les composent. Les arcs présentent des connexions filaires et possèdent une capacité de transfert maximale de données définie par une constante.

Expérimentations et résultats de l'évaluation du modèle : les différentes simulations sont effectuées avec un PC contenant des processeurs Intel(R) Core (TM) i5-3360M avec 4 cœurs, une puissance de 2.8 GHz par cœur et 4 Gb de mémoire RAM. On utilise le solveur CPLEX version

Catégorie	CPU node	RAM (Gb)	SSD (Go)	CPU freq per core (GHZ)	Bdw (GB)	Slots Map	Slots Reduce
c3.2xlarge	8	15	160	2.8 Intel Xeon E5-2680v2	1	5	2
i2.2xlarge	8	61	1600	2.5 Intel Xeon E5-2670v2	1	4	3
r3.xlarge	4	30.5	1600	2.5 Intel Xeon E5-2670v2	2	2	1

TABLE 1 – Configuration des machines utilisées pour la génération de données du problème

	N1	N2	N3	M1	M2	M3	Blocs	N	M
Scén 1	1	1	1	1	2	0	10	25	2
Scén 2	1	2	1	2	1	0	10	33	3
Scén 3	0	0	3	0	2	0	10	36	2
Scén 4	2	2	1	1	2	1	10	38	4
Scén 5	4	1	1	1	2	1	10	40	4
Scén 6	2	3	1	0	2	0	10	46	2

TABLE 2 – Scénarios de la génération des données d'entrées lors de l'évaluation du problème

Type des travaux	$n_i^r = n_i^b * S * XY$	$n_i^h = (n_i^b * S * WZ)/1024$
(1)	$XY \in [0.3, 0.6]$	$WZ \in [13, 26]$
(2)	$XY \in [0.4, 0.8]$	$WZ \in [30, 46]$
(3)	$XY \in [0.6, 1]$	$WZ \in [46, 76]$

TABLE 3 – Formules utilisées pour la génération des quantités de mémoire et de disque pour chaque tâches

12.4 pour évaluer le modèle présenté, CPLEX est configuré (i) pour utiliser 2 Gb de mémoire RAM, (ii) pour effectuer les traitements en mode parallèle (4 processus) et (iii) une limite de temps égale à 1800 secondes. Si la limite de temps ou de la mémoire est atteinte pour une instance donnée, la solution de cette dernière sera déclarée non résolue. Initialement l'horizon de temps T est calculé de la façon suivante : $T = \frac{\sum_{i \in L^r} P_i}{\sum_{j=0}^{M} m_j^{S^r}} + \frac{\sum_{i \in L^m} P_i}{\sum_{j=0}^{M} m_j^{S^m}} + 2$. Si pour une instance donnée d'un scénario, le modèle n'a pas trouvé de résultat, l'horizon de temps est incrémenté de 1 pour l'itération suivante. Les différents scénarios qu'on a choisis pour l'évaluation du modèle sont présentés dans le tableau 2, de la colonne 2 à la colonne 4, le nombre de travaux pour les différentes tailles ; de la colonne 5 à la colonne 7, le nombre de machines pour les différentes catégories de machine ; la huitième colonne désigne le nombre de blocs de données par scénario. Les deux dernières colonnes présentent le nombre total de tâches et de machines par scénario. On considère trois tailles de travaux : $(N1)$ travaux non chargés avec 5 tâches dont 2 tâches "Map" et 3 tâches "Reduce". $(N2)$ travaux sous chargés avec 8 tâches dont 2 tâches "Map" et 6 tâches "Reduce". $(N3)$ travaux chargés avec 12 tâches dont 3 tâches "Map" et 9 tâches "Reduce". Les résultats d'évaluation du modèle sont affichés dans le tableau 4, les colonnes désignent respectivement : (1) les scénarios, (2)le nombre d'instances que le solveur n'a pas pu résoudre, (3) le nombre d'instances résolues, (4) le nombre d'instances pour lesquelles CPLEX s'arrête par-ce qu'il a atteint la limite mémoire, (5) le nombre d'instances pour lesquelles CPLEX s'arrête par-ce qu'il a atteint la limite de temps fixé, du (6) au (8) le nombre (minimal, maximal et moyen) de nœuds explorés par CPLEX durant la résolution du problème avec la méthode par séparation et évaluation et enfin, du (9) au (12) le temps d'utilisation du processeur pour trouver les différentes solutions.

	Sol. non résolus	Sol. résolus	Lim Mémoire	Lim Temps	N_{min}	N_{moy}	N_{max}	T_{min}	T_{moy}	T_{max}
Sc 1	0	10	0	0	547	1689.8	8488	0	1.4	6
Sc 2	0	10	0	0	876	4303.9	11061	1	9.2	28
Sc 3	0	10	0	0	72	514.3	4136	1	2.1	10
Sc 4	0	10	0	0	2113	22160.3	127413	6	121.7	969
Sc 5	0	9	1	0	9492	22213.8	46759	33	93.22	295
Sc 6	0	10	0	0	9336	28298.2	52746	72	435	916

TABLE 4 – Résultats des calculs (10 instances par scénario)

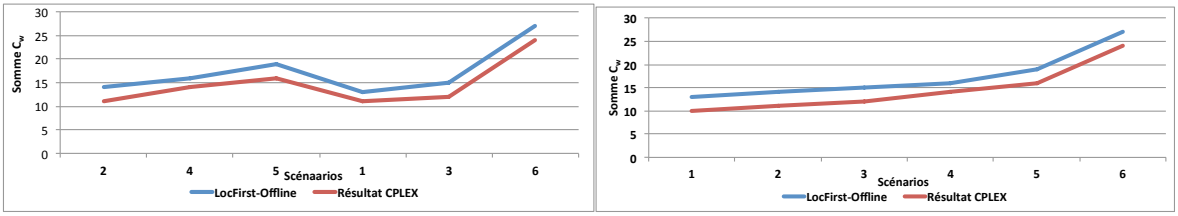
4. Présentation et évaluation de l'heuristique de résolution pour la version hors ligne

L'heuristique de résolution proposée est nommée "LocFirst" (algorithme 1), elle est basée sur les algorithmes de listes, elle prend en entrée : la liste des travaux, la liste des tâches, la liste des machines et la liste des blocs.
Présentation de l'heuristique : on commence (ligne 2) par trier la liste des machines "Lmachines" et la liste des tâches "Ltasks" de façon décroissante en fonction de la quantité des ressources : pour les machines, on trie en fonction de la quantité de slots, si on a la même quantité de slot, on trie en fonction de la quantité de mémoire RAM disponible sinon en fonction de l'espace existant sur le disque dur. Ensuite, on trie la liste des tâches en fonction de la quantité RAM et espace disque dur qu'ils consomment. L'ordonnancement des tâches est constitué de deux niveaux, lorsqu'on a des ressources libres, on commence par ordonnancer les tâches sur les machines contenant les données qu'elles traitent. Ensuite, on ordonnance les tâches sur les ressources restantes en commençant par les machines contenant le plus de ressources. Pour le premier niveau (ligne 3), pour chaque travail w de la liste "LTravaux" et pour chaque tâche qui lui est associée de la liste "Ltasks", on récupère la liste des machines "ListMbloc" contenant les données qu'elle traite et leurs réplications (ligne 5). On parcourt "ListMbloc" et on affecte la tâche T_i^w à la première machine $m^{'}$ possédant les ressources pour l'exécuter. La récupération de la liste des machines contenant les données consiste à : (i) parcourir "LdataBloc" pour récupérer l'emplacement des données, ensuite, (ii) parcourir "Lmachines" et insérer les machines dans "ListMbloc", de ce fait, les machines dans la liste "ListMbloc" sont triées de façon croissante en fonction des ressources. Pour le deuxième niveau (ligne 10), les tâches non ordonnancées sont affectées sur les ressources restantes et qui peuvent exécuter des tâches. On récupère chaque travail w de la liste "LTravaux", pour chaque tâche T_i^w de liste "Ltasks", on parcourt "Lmachines" et on affecte la tâche T_i^w à la première machine $m^{'}$ possédant les ressources pour l'exécuter. Après la fin des tâches associées, le travail est enlevé de la liste "LTravaux" et les tâches associées sont enlevées de "Ltasks".

Algorithme 1 : Algorithme associé à l'heuristique LocFirst

Données :
1 $Lmachines \leftarrow \{m\}, \forall m = 1...M$ $LdataBloc \leftarrow \{b\}, \forall b \in A^b$ $Ltasks \leftarrow \{t\}, \forall t = 1...N$ $LTravaux \leftarrow w^{'}, \forall w^{'} = 1...W$
Résultat : Les tâches de Ltasks sont ordonnancées sur les machines de la liste Lmachines

2 `TrieLists(`*Lmachines, Ltasks*`)` ;
3 **pour chaque** *travail* $w \in LTravaux$ **faire**
4 **pour chaque** *tâche* $T_i^w \in Ltasks$ **faire**
5 $ListMbloc \leftarrow$ `RetounerListeMachineBloc(`$T_i^w, LdataBloc, Lmachines$`)`
6 **pour** $m' \in ListMbloc$ **faire**
7 **si** $m^{'}$ *contient les ressources nécessaires pour exécuter* T_i^w **alors**
8 `ExecuterTâche(`T_i^w,$m^{'}$`)` ;
9 $Ltasks \leftarrow Ltasks \setminus \{T_i^w\}$;
10 **pour chaque** *travail* $w \in LTravaux$ **faire**
11 **pour chaque** *tâche* $T_i^w \in Ltasks$ **faire**
12 **pour** $m^{'} \in Lmachines$ **faire**
13 **si** $m^{'}$*contient les ressources nécessaires pour exécuter* T_i^w **alors**
14 `ExecuterTâche(`T_i^w, m'`)`;
15 $Ltasks \leftarrow Ltasks \setminus \{T_i^w\}$;
16 **si** *toutes les tâches associées à un travail* w' *sont ordonnancées* **alors**
17 $LTravaux \leftarrow LTravaux \setminus \{w^{'}\}$; $Ltasks \leftarrow Ltasks \setminus \{T_i^{w^{'}}\}$;



(a) Scénarios ordonnés selon le degré de parallèlisme (b) Scénarios ordonnés selon le nombre de tâches

FIGURE 1 – Évaluation de l'heuristique : "FLocFirst-Offline"

Évaluation de l'heuristique : on compare dans cette partie la moyenne des résultats fournis par l'heuristique "LocFirst" à la borne inférieure calculée (la moyenne des solutions optimales trouvées par CPLEX sur 10 instances de chaque scénario). On utilise la même politique décrite dans (section 3) pour la génération des entrées de l'heuristique. On utilise les mêmes scénarios décrits dans le tableau 4. La figure 1 présente les résultats de l'évaluation. Elle est composée de deux sous figures : dans (a), l'axe des abscisses présente les différents scénarios ordonnés de façon croissante selon le degré de parallélisme (nombre totale de cœurs virtuel ou slots) utilisés par chaque scénario. L'axe des ordonnées présente la valeur moyenne de la fonction objective. La sous figure (b) présente les mêmes résultats que la précédente cependant les scénarios sont ordonnancés selon le nombre de tâches qu'ils contiennent. On remarque que sur des instances de petite taille, l'heuristique fournit de bons résultats et le ratio entre les résultats fournis par l'heuristique et les résultats fournis par le solveur est presque constant (environ 7%). Le temps utilisé par l'heuristique "LocFirst" pour le calcul de la solution est de l'ordre de la milliseconde pour tous les scénarios utilisés dans la comparaison.

5. Conclusion

Le problème d'ordonnancement dans le logiciel Hadoop est un problème avec des contraintes de ressources, les tâches sont non interruptives et de durée variable, une relation de précédence est définie entre les différentes tâches. On modélise le problème en utilisant la modélisation mathématique indexée sur temps. On évalue le modèle proposé et on calcule une borne inférieure de référence, on utilise le solveur CPLEX pour le calcul de la solution exacte. On propose une heuristique dont on évalue les résultats par rapport à la borne inférieure calculée précédemment. Ce travail nous a permis (i) de présenter le problème d'ordonnancement dans Hadoop et de le modéliser, (ii) de proposer une première heuristique et de l'évaluer. Vu la complexité du modèle et les limites présentées par le solveur CPLEX, on a pu évaluer l'heuristique sur des instances de taille moyenne. L'heuristique proposée fournit des bons résultats sur les instances utilisées, le ratio entre les résultats qu'elle fournit et la borne inférieure calculée est contant (environ 7%) avec un temps de calcul de l'ordre de la milliseconde. Dans Hadoop, l'ordonnanceur n'a aucune information concernant ni la date de soumission des travaux ni l'état de la grappe. De ce fait, la prochaine étape consiste à étudier le problème dans un contexte en ligne et sur des instances de grandes tailles.

Bibliographie

1. C. (S.), Kasiviswanath (N.) et Reddy (P. C.). – Article : A survey on big data management and job scheduling. *International Journal of Computer Applications*, vol. 130, n13, November

2015, pp. 41–49. – Published by Foundation of Computer Science (FCS), NY, USA.

2. Dean (J.) et Ghemawat (S.). – Mapreduce : Simplified data processing on large clusters. – In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, *OSDI'04*, OSDI'04, pp. 10–10, Berkeley, CA, USA, 2004. USENIX Association.
3. Foundation (T. A. S.). – Hadoop : Capacity scheduler, March 2016.
4. MAEDEH (M.), SAFI (E. F.) et H. (N. M.). – A comprehensive view of hadoop research—a systematic literature review. *Journal of Theoretical and Applied Information Technology*, vol. 66, n3, 2014, pp. 661–670.
5. Polato (I.), Ré (R.), Goldman (A.) et Kon (F.). – A comprehensive view of hadoop research—a systematic literature review. *Journal of Network and Computer Applications*, vol. 46, 2014, pp. 1 – 25.
6. Sandholm (T.) et Lai (K.). – Dynamic proportional share scheduling in hadoop. – In *Proceedings of the 15th International Conference on Job Scheduling Strategies for Parallel Processing*, *JSSPP'10*, JSSPP'10, pp. 110–131, Berlin, Heidelberg, 2010. Springer-Verlag.
7. Tao (Y.), Zhang (Q.), Shi (L.) et Chen (P.). – Job scheduling optimization for multi-user mapreduce clusters. – In *Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on*, pp. 213–217, Dec 2011.
8. Zhou (W.), Han (J.), Zhang (Z.) et Dai (J.). – Dynamic random access for hadoop distributed file system. June 2012, pp. 17–22.